Institut d'Intelligence Artificielle

Université Paris VIII

NTP N° 2.

MANUEL LISP 510

DESCRIPTION et UTILISATION

INTRODUCTION

Le système LISP 510 a été implémenté sur l'ordinateur CAE 510 en Juin 1972 par P. GREUSSAY.

Le systéme est destiné aux recherches en Intelligence Artificielle effectuées au département d'Informatique de l'Université PARIS VIII.

Les données d'applications sont principalement :

- composition musicale et plastique ;
- démonstration automatique de théorèmes en logique modale et temporelle ;
- simulation d'entretien en langue naturelle, à orientation psychiatrique ;
- conception et mise au point de systèmes de programmation non déterministes et heuristiques.

L'implémentation était, à l'origine, conçue comme une version simplifiée du système LISP BBN [4,6], si possible portable [7], et macrogénérée au moyen du système STAGE 2 [8], adapté sur CAE 510 à cet effet. Il a été très vite évident que ni la taille de la mémoire disponible (8192 mots de 18 bits), ni la vitesse de calcul de la machine, ne permettraient cette simplification. Pour les mêmes raisons d'insuffisance de mémoire, et d'absence de mémoire auxiliaire rapide, on a dû renoncer à la possibilité d'adjoindre un compilateur au système. Le système se présente donc comme un interprète, extrêmement proche d'un macrogénérateur et en possède les inconvénients (lenteur relative) et avantages (capacité d'auto-modification, traitement «propre» des variables libres, mode conversationnel).

Cette brochure est une description complète du maniement du système LISP 510. Elle n'a en aucun cas la prétention d'être une introduction au langage LISP, elle suppose au contraire que le lecteur en a déjà une certaine connaissance.

Pour s'y initier, on consultera [1] ou [5], [2] décrivant plutôt la logique et l'interprétation du langage, le classique [9] n'étant guère indiqué aux débutants. Il faut mentionner que le recueil d'articles [3] contient un excellent ensemble d'exercices en «LISP pur» accompagnés de leurs solutions.

Pour faciliter la tâche de l'utilisateur, le système LISP 510 comprend plusieurs programmes de TRACE des fonctions LISP, un éditeur, et un petit prélude permettant de changer aisément, lors de l'initialisation du système, les dimensions des piles de travail et de la liste libre.

Le système permet la lecture des fonctions

- sur cartes perforées (en mode batch)

- sur console (en mode conversationnel)

et l'écriture des résultats

- sur console

- sur table traçante. (à option).

BIBLIOGRAPHIE

[1] W.D. MAURER : The programmer's introduction to LISP 1972, Elsevier

[2] D. RIBBENS : Programmation non numérique LISP 1.5 1969, Dunod

[3] BERKELEY and BOBROW, eds : The programming Language LISP : Its Operation and Applications 1964 MIT Press

[4] BOBROW, MURPHY : Structure of a LISP System Using two-level Storage. Mars 1967, CACM

[5] WEISSMAN : LISP 1.5 Primer 1967, Dickenson Publishing Company

[6] TEITELMAN, BOBROW, HARTLEY, MURPHY : BBN - LISP . TENEX Reference Manual 1971, Bolt Beranek and Newman.

[7] NORDSTROM, SANDEWALL, BRESLAW : LISP FI : a Fortran implementation of LISP 1.5 1970, Uppsala University

[8] W.M. WAITE : The Mobile Programming System : STAGE 2. Juillet 1970, CACM

[9] Mc CARTHY et al. : LISP 1.5 Programmer's Manual 1965 , MIT Press

[10] WEIZENBAUM : ELIZA. A Computer Program for the Study of Natural Language Communication between Man and Machine. Janvier 1966, CACM

[11] COLBY : Artificial Paranoia
Artificial Intelligence - 2 (1971)

[12] BOUSSARD, PAIR : Introduction à ALGOL 68 , RIRO 1969 nº B - 3

[13] Mc CARTHY : Recursive Functions of Symbolic Expressions and their Computations by Machine. Avril 1960 , CACM

PRELUDE

Après le chargement de LISP

- sur le lecteur de ruban perforé
- sur le dérouleur de bande magnétique n° 1

le système attend une commande ; on transmettra sur la machine à écrire une des commandes que voici, suivie d'un retour à la ligne.

LISP — Le prélude donne la main à LISP 510 proprement dit, et ne pourra plus être désormais récupéré, étant chargé dans la zone des piles de travail.

PRINT — Le prélude imprime un deck de cartes, se terminant par une carte comportant NIL cadré à gauche (i.e. à partir de la première colonne). L'impression est alors limitée à la *zone de texte* sur la carte, permettant ainsi de ne pas trop dépendre de la lenteur de la MAE. Après la lecture de la carte NIL, le prélude attendra une nouvelle commande.

SEE — Le système informe l'utilisateur des adresses standard des piles de travail (liste libre, piles, zone atomes CF. Fig I) permettant de juger de l'utilité d'une (ou plusieurs) modifications.
ex : permettre un plus grand nombre d'atomes, ou une pile plus profonde, ou un plus grand nombre de doublets, etc ... Le prélude imprime, n étant un entier décimal.

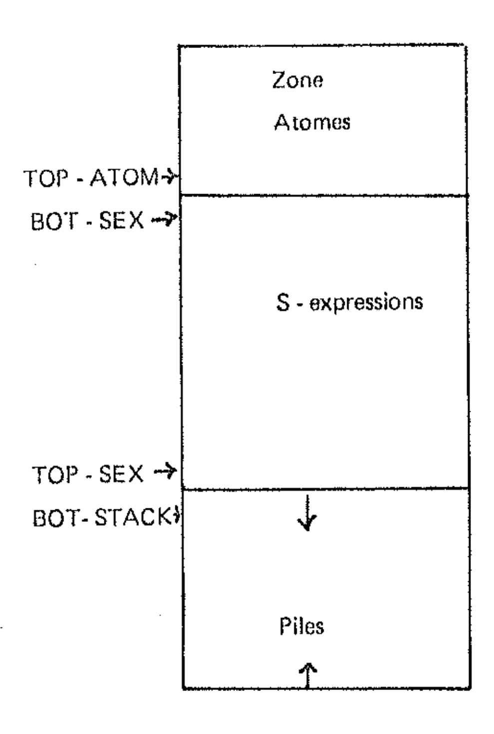
1.	TOP - ATOM	n	: adresse maximum d'atome
2.	BOT - SEX	n	: adresse de base de liste libre (en principe TOP - ATOM + 4)
3.	TOP - SEX	n	: adresse du plus haut doublet en liste libre.
4.	BOT - STACK	n	; adresse de base de la pile de travail (en principe TOP-SEX+ 2).
5..	LEN - BUF	n	: nombre de caractères d'une ligne d'impression sur MAE.

L'entier (1, 2, 3, 4 ou 5) précédant l'indicateur servira éventuellement de préfixe à la commande de modification PUT des adresses en question.

PUT — Pour modifier les adresses visualisées par SEE.
La commande PUT étant frappée, on poursuivra en frappant une *sous-commande* de format que voici : (sachant que n = 1, 2, 3, 4 ou 5 correspond à l'indicateur dans SEE, et e = un entier interprétable comme une adresse de mot de CAE 510)
n 2 espaces e
et l'adresse n sera alors initialisée à l'entier e.

On refrappera cette sous-commande autant de fois que nécessaire, et on terminera par tout caractère différent de 1, 2, 3, 4 ou 5 pour revenir au prélude.

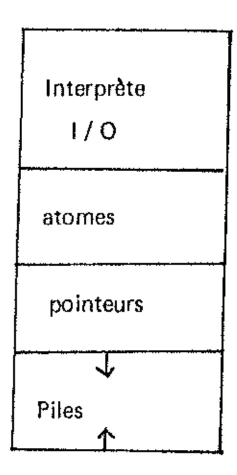
FIG. I



Une fois toutes les modifications affectuées, l'utilisateur frappera la commande LISP, qui lui livrera un système taillé sur mesure. On notera que le système n'éffectue pas de garbage collecting d'atomes. On aura par conséquent intérêt, en particulier pour le traitement d,énoncés en langage naturel [10], [11], à dépasser largement le nombre standard d'atomes-usager permis (en principe une centaine), au moyen de la commande PUT.

DONNEES . ORGANISATIONS ET TYPES.

Voici, après le chargement du système LISP 510, la configuration de la mémoire :



En dehors de l'interprète proprement dit et des piles de travail, une certainequantité de mémoire est assignée d'office aux typres de données particuliers.

En LISP 510, on aura affaire à 2 types de données :

- pointeurs
- atomes

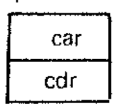
auxquels serort affectés deux zones de mémoire distinctes. Sauf modifications (CF. PRELUDE) le nombre initial disponible d'atomes définissables par l'utilisateur ne sera pas inférieur à 100, et le nombre de doublets ne sera pas inférieur à 1750.

POINTEURS : Ils occupent un mot de CAE 510. Le mot contiendra l'adresse d'un atome ou celle d'un autre pointeur.

Un *DOUBLET* sera composé de deux pointeurs physiquement consécutifs.

Une *LISTE* est une chaîne de doublets, non nécessairement contigus.

LISTES : Pour toute liste, la partie CAR pointera sur le premier élément de la liste, et la partie CDR pointera sur le doublet suivant de la liste.



Du point de vue externe, une liste est une suite d'au moins un élément — ( ) est lu comme représentant l'atome NIL — i.e. atome literal, nombre, liste, qui sera inscrite entre une parenthèse ouvrante, et une parenthèse fermante.

Si la liste comporte deux éléments ou plus, l'élément final peut être précédé de «.» (DOT), indiquant que le CDR du dernier doublet de la liste est l'élément suivant immédiatement le «.».

EX : (A . B) ⟷ [A | B]

(A B . C) ⟷ [A | •]→[B | C]

S'il n'y a pas d'occurence du «.», le CDR du dernier doublet d'une liste est implicitement NIL.

Les listes sont construites par les fonctions primitives CONS et LIST (et par READ naturellement).
Les listes seront imprimées dans l'ordre que voici :

- une parenthèse ouvrante
- le premier élément de la liste
- un espace
- le second élément

etc ...

jusqu'à ce que le doublet final soit atteint.
L'impression des listes est considérée comme achevée lorsque le CDR d'un doublet n'est pas une liste. Si le CDR d'un doublet final est NIL, sont alors imprimés consécutivement le CAR de ce doublet et une parenthèse fermante. Si, au contraire le CDR d'un doublet final *n'est pas* NIL, on aura alors l'impression du CAR de ce doublet, suivi d'un «.», puis du CDR de ce doublet, suivi enfin d'une parenthèse fermante.

EX :	*LISTE LUE*	*LISTE IMPRIMEE*
	(A B C . NIL)	(A B C)
	(A B . C)	(A B . C)

L'occurence d'un «.» dans une liste lue ne peut avoir lieu qu'en position PRECEDANT le dernier élément (externe). Ceci sera considéré ou non comme une restriction de LISP 1.5.
Sachant que :

(A B . C D) est sans signification

et

(A . (B C)) est équivalent à (A B C)

ATOMES : ***Le point de vue externe :***

On considérera comme ***atome*** une chaîne de caractères de longueur quelconque, ne comportant aucune occurence d'un des caractères suivants :

espace ( ) . ;

seront considérés comme séparateurs les caractères que voici :

espace ( ) .

Sera considéré comme un ***commentaire*** une chaîne de caractères quelconques, chaîne de longueur quelconque comprise entre deux occurences de «;».

EX : AT ; COMMENT ... ( ) ; M1 équivaut à la donnée de l'atome «ATM1».

Seuls les *SIX* premiers caractères ou plus de la chaîne seront effectivement pris en compte.

On fera dès l'abord la distinction entre

- atomes literals, i.e. non interprétables comme des nombres
- atomes numériques.

Le point de vue interne :

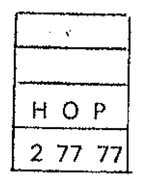
ATOMES LITERALS

Un atome litéral est une zone de quatre mots de mémoire consécutifs

α	
β	
γ_1	
γ_2	

Le P-NAME de l'atome est stocké en $\gamma 1$ - $\gamma 2$, à raison de trois caractères par mot. Si le P–. NAME comporte moins de six caractères, les zones innocupées seront remplies de caractères non imprimables : 77 octal

EX : HOP2

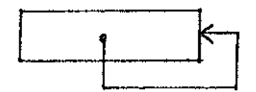


ZONE α *:* elle contiendra, pour les fonctions standard l'adresse du sous programme en langage machine correspondant. Celles-ci ne comporteront donc pas d'indicateurs SUBR où FSUBR comme en LISP 1.5

Le contenu de la zone sera désigné dans tout ce qui suit sous le nom de ***C-VALEUR*** (cell-value) de l'atome.
En ce qui concerne les atomes non-standard (i.e. introduits par l'utilisateur), la zone sera *indéfinie* lors de la définition de l'atome literal. La modification de cette zone pourra être effectuée par l'usage de la fonction RPLACA.

Il est à noter que indéfini *, qui n'est pas une valeur pour l'utilisateur,en est une pour l'interprète. En effet, lors de l'évaluation d'un atome, l'interprète cherchera d'abord s'il lui correspond une C-VALEUR distincte de* indéfini *; s'il en est ainsi, cette C-VALEUR sera la valeur de l'atome. Dans le cas contraire, et seulement dans ce cas, l'interprète cherchera si l'atome possède une valeur sur la A-liste (A-valeur). Si un atome est lié sur la A-liste, on se gardera bien de lui donner une C-valeur, toute référence ultérieure à la A-liste devenant alors sans effet : la valeur* indéfini *étant, comme quelques autres, une qualité qu'on ne perd qu'une fois.*

Pour les atomes standard que voici : NIL, T, EXPR, FEXPR, LAMBDA, LABEL, FUNARG la C-VALEUR sera un pointeur sur ces atomes.



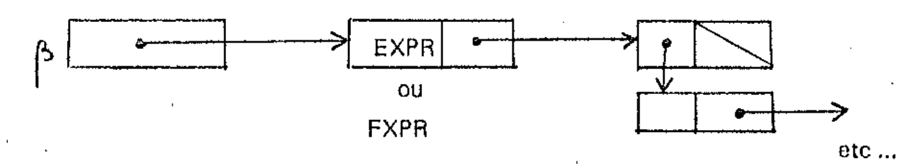
On retiendra par conséquent que
(CAR NIL) = (CDR NIL) = (CAAR NIL) = (CADDAR NIL) = ... = NIL

ZONE β : elle est, pour les atomes standard, initialisée à NIL, à l'exception de celle de l'atome REM, qui contiendra, à tout instant le nombre de doublets libres dégagés par le dernier garbage collecting.

L'interprète est construit de telle sorte que l'utilisateur peut, si cela lui paraît utile, redéfinir en EXPR ou FEXPR des fonctions standard, au moyen de DE, DF ou DEFINE. L'atome acquiert alors le statut d'une fonction - utilisateur, tant que l'indicateur EXPR ou FEXPR reste présent sur la liste, sur laquelle pointe alors le contenu de la zone β *. Cette transformation est reversible : la suppression de cet indicateur restitue alors à cet atome son statut initial de fonction standard.*

EX : LENGTH, qui calcule le nombre d'éléments de plus haut niveau d'une liste, pourra être, à l'occasion, redéfini comme une EXPR calculant le nombre d'atomes DISTINCTS que comporte une liste, quel que soit le niveau de son occurence.

S'il s'agit d'un atome - fonction défini par l'utilisateur on aura (par DE ou DF ou DEFINE)



Les modifications de la zone β sont effectuables en employant la fonction RPLACD.

En LISP 510, on distinguera donc pour tout atome litéral .

- une C - VALEUR : contenu de la zone α
- une P - liste : contenu de la zone β
- un P - NAME : contenu des zones $\gamma 1 - \gamma 2$

Une telle structure de quatre mots consécutifs en zone de mémoire ***atomes*** est créable au moyen des fonctions READ, NEXTCH et GENSYM. Quelle que soit le nombre d'occurences distinctes du même P - NAME dans un programme LISP, seule la premiè-re occurence induira la création d'un atome litéral interne, toutes les occurences suivantes pointeront désormais sur la structure ainsi crée. Un atome litéral aura donc une *représentation interne unique.*

On notera de surcroît qu'aucun garbage collecting n'est effectué sur les atomes litérals.

ATOMES NUMERIQUES

Ils offrent l'apparence externe de la représentation conventionnelle d'entiers, précédés ou non d'un signe. Contrairement aux atomes litérals, ils sont stockés directement dans les structures de listes dont ils sont éléments, i.e. en zone pointeur. on pourra par conséquent avoir simultanément en mémoire plusieurs occurences physiquement distinctes du même atome numérique. Ils seront distingués des *adresses* proprement dites par l'adjonction automatique d'une constantes (en l'occurence 60 000). Il va de soi que tout se passe comme si, pour l'utilisateur, cette adjonction n'avait pas lieu. A un entier sera donc alloué la même place qu'un pointeur, à savoir un mot de CAE 510.

En conséquence, un atome numérique

- ne comportera pas de P - LISTE
- ne comportera pas de C - VALEUR
- ne possèdera pas de P - NAME à proprement parler, son impression étant alors assurée de façon spécifique par les fonctions PRINT et PRIN1.

On pourra, selon l'ontologie particulière de l'utilisateur considérer qu'un atome numérique est

- un pointeur sur lui même
  (ce qui est substanciellement faux)
- sa propre valeur
  (ce qui est formellement faux).

Un atome numérique est créé lors d'un READ, ou comme résultat de l'évaluation d'une expression arithmétique, ou comme valeur ramenée par l'évaluation d'une fonction à résultat numérique.

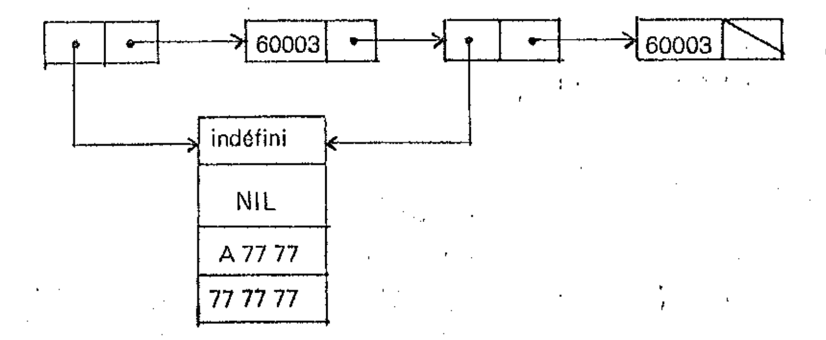
Il est sans doute regrettable que LISP 510 ne traite pas les nombres flottants. Dans tous les cas, on s'abstiendra d'utiliser des entiers non inscrits dans l'intervalle

– 50 000 , + 50 000

EX : soit la liste lue :

(A 3 A 3)

et sa représentation interne



TYPES DES FONCTIONS

En LISP 510, on peut distinguer les types de fonctions comme suit :

a. les arguments seront ou ne seront pas évalués

b. elles comporteront un nombre fixeou variable d'arguments

c. elles seront définies par uneexpression LISP, ou sont directement écrites en langage machine.

En théorie, on disposerait de 8 types distincts (2 x 2 x 2). Toutefois en LISP 510, ces caractéristiques ne sont pas strictement indépendantes, et on ne distinguera que 5 types distincts, sous les noms que voici :

EXPR
FEXPR
SUBR
FSUBR
SUBR - N

EXPR

Les fonctions définies par des expressions LISP sont appelées des *exprs.*Elles seront définies au moyen des fonctions DE ou DEFINE. Tous les arguments, en nombre *fixe*, seront évalués.

FEXPR

Il s'agit également de fonctions définies par des expressions LISP. Elles seront définies au moyen de la fonction DF. Elles comporteront un nombre *quelconque* d'arguments qui ne *seront pas évalués.*

Les exprs et les fexprs seront donc des fonctions définies par l'utilisateur. Leur usage sera cependant fort distinct. Les exprs correspondent à la notion traditionnelle de fonctions : les arguments actuels sont évalués, puis liés sur la A-liste avec les arguments formels, puis l'interprète passe la main au *corps* de la fonction. Donc dans le cas des exprs les arguments sont appelés par valeur. Le cas des fexprs est fort différent, et correspond de façon lointaine à la notion d'appel par nom. Les fexprs ne comprendront que 2 arguments formels, même si à l'appel, le nombre d'arguments actuels est quelconque. En effet, à l'appel d'une fexpr, au premier argument sera liée (sur la A-liste) la liste des arguments actuels non évalués, et la nouvelle A-liste ainsi obtenue, sera liée au second argument. Ce dernier est donc accessible dans le corps de la fexpr, et donne par conséquent au programmeur l'opportunité de modifier ou de consulter directement la A-liste.

exemple d'utilisation : la fexpr PR - AT imprime sur la même ligne tous ses arguments :

```
(DF PR - AT (E A)
        (MAPC E (QUOTE PRIN1 ) ) )
```

SUBR

Les subrs sont écrites en langage machine et correspondent au exprs.

FSUBR

Les fsubrs sont écrites en langage machine et correspondent aux fexprs.

SUBR - N

Les subr - ns sont écrites en langage machine. Elles prendront un nombre d'arguments quelconque qui seront tous évalués.

En général on notera les caractéristiques suivantes :

- 1 / l'évaluation des arguments a lieu de gauche à droite, on en déduira les effets de bords.

- 2 / en pratique, *toutes* les fonctions peuvent, à l'appel, comporter un nombre d'arguments quelconque. Reste que dans le cas des fonctions à nombre fixe d'arguments, les arguments supplémentaires (à l'appel) seront simplement évalués, et non liés, et pour cause, à des arguments formels absents. Si, au contraire des arguments font, à l'appel, défaut, ceux-ci seront liés à NIL sur la A-liste. (voir en appendice la « Note sur la A-liste »).

- 3 / quant à la représentation interne, créée par l'appel des fonctions DE et DF, on la déduira des utilisations équivalentes de RPLACD que voici :

```
(DE  FOO  liste-args  corps)
          (RPLACD (QUOTE  FOO)
                  (QUOTE (EXPR (LAMBDA  liste-args
                                        corps))))
(DF  FOO  liste-args  corps)
          (RPLACD  (QUOTE FOO)
                  (QUOTE (FEXPR (LAMBDA  liste-args
                                        corps))))
```

- 4 / Dans le cas des EXPRs, la liaison des arguments actuels évalués aux arguments formels obéit à la logique de la fonction PAIR. Par conséquent une liste d'arguments formels peut avoir la forme :

(A1 A2 . A3)

On tient dans ce cas une forme spéciale de fonction à nombre quelconque d'arguments. Les valeurs des deux premiers arguments seront liées respectivement à A1 et A2, la liste des valeurs des arguments restant étant alors liée à A3. On pourra de même définir des fonctions de la forme :

(DE FOO A corps)

Sur la A-liste, à l'appel, A sera lié à la liste des valeurs des arguments.

exemple d'utilisation : la fonction PROG1, à nombre quelconque d'arguments évalue tous ses arguments, et ramène la valeur du premier :

(DE PROG1 U (CAR U))

- 5 / Seules les exprs et les fexprs comportent les indicateurs EXPR et FEXPR sur leurs P-listes. Il n'existe pas, en LISP 510 d'indicateur SUBR et FSUBR, les fonctions standard étant ici directement accessibles à l'interprète.

FONCTIONS ET PREDICATS PRIMITIFS

CAR, CDR, CAAR, CADR, CDAR, CDDR, CADDR, CONS, RPLACD, RPLACA, QUOTE, COND, PROG, PROGN, GO, GOTO, RETURN, SET, SETQ, ATOM, NUMBP, EQ, NULL, NOT, EQUAL, AND, OR, MEMBER.

FONCTIONS PRIMITIVES

(CAR X) *car* livre le premier élément d'une liste X ou l'élément de gauche d'une paire pointée
- SUBR - 1 X. Pour les atomes litteraux, le résultat est indéfini.

(CDR X) *cdr* livre le reste de la liste (la liste X moins le 1er élément). C'est aussi l'élément droit
- SUBR - 1 d'une paire pointée.

pour obtenir la P - liste d'un atome A, on évaluera (CDR (QUOTE A)).

(CAAR X) = (CAR (CAR X))
(CADR X) = (CAR (CDR X))

.
.
.

(CADDR X) = (CAR (CDR (CDR X)))
- SUBR - 1

(CONS X Y) *cons* construit une paire pointée (X . Y). Si Y est une liste, X devient le 1er
- SUBR - 2 élément de cette liste. Le résultat est indéfini si Y n'est pas une liste.

(RPLACD X Y) Place le pointeur Y dans la partie CDR du doublet pointé par X. La struc-
- SUBR - 2 ture interne de X est physiquement modifiée, contrairement à CONS qui créé un nouvel élément de liste. La valeur deRPLACD est X .

– RPLACD est avec NCONC le seul moyen de créer des listes circulaires (donc sans représentation externe). Cet accès à la structure physique des doublets est un peu une entorse à la définition du « LISP pur». RPLACD est utilisé par DE, DF et DEFINE pour la définition de fonctions.

Ex : création de fonctions synonymes :

« SECOND » pour CADR :

(RPLACD (QUOTE SECOND) (QUOTE (EXPR CADR)))

(RPLACA X Y) semblable à RPLACD mais remplace la partie CAR de X par Y. La
- SUBR - 2 valeur de RPLACA est X.

*RPLACA et RPLACD modifiant physiquement la structure de liste de X seront donc souvent utilisées pour l'*AUTO - MODIFICATION *de fonctions par apprentissage. On s'attendra cependant à des effets étranges, lors de la mise au point (en particulier dans le cas des RPLAC(A+D)*

accidentels sur les atomes systèmes).

(QUOTE X) - FSUBR — Cette fonction inhibe l'évaluation de son argument. Sa valeur est X lui-même.

(COND C1 C2 ... CK) - FSUBR — La fonction conditionnelle de LISP, *cond*, prend un nombre quelconque d'arguments C1, C2, ..., CK, appelés *clauses*. Chaque clause Ci est une liste ($E1_i$ $E2_i$). Les clauses sont considérées en séquence comme suit : la première expression $E1_i$ de la clause Ci est évaluée et sa valeur est classifiée comme *fausse* (égale à NIL) ou *vraie* (non égale à NIL). Si $E1_i$ est *vraie*, l'expression $E2_i$ est évaluée et la valeur de COND est la valeur de $E2_i$, la seconde expression de la clause.
Si $E1_i$ est *fausse*, alors le reste de la clause C_i est ignoré et la clause suivante C_{i+1} est considérée.
Si aucun $E1_i$ n'est *vrai*, la valeur de l'expression conditionnelle est NIL.

(PROG ARG1 E1 E2 .... En) - FSUBR — Cette fonction permet d'écrire un programme en style - ALGOL
Le 1er argument est une liste des variables internes au programme (ou NIL si aucune variable n'est nécessaire). Chaque atome de cette liste est initialement lié à NIL. Cette liste sera de la forme : (AT1 AT2 ... ATn) ou ( ). Le reste du *PROG* est une séquence d'énoncés non atomiques (à évaluer) et d'atomes (littéraux et numériques) utilisés comme étiquettes pour *GO* et *GOTO*. Les énoncés sont évalués séparément, en sautant les étiquettes. Les 3 fonctions *go, goto* et *return* modifiant cette séquence. La valeur du PROG est en général spécifiée par la fonction *RETURN*. Si le programme épuise toutes ses instructions sans *return*, sa valeur est alors celle de la dernière instruction évaluée.

Les étiquettes multiples sont permises.

Ex ; *si N = 0, 1, ..., 5 et l'instruction suivante doit être S1 si N est pair, S2 sinon, on économisera l'usage de* REM *par : (GOTO N)*

0 2 4 (S1)

1 3 5 (S2)

Les étiquettes étant placées dans une liste spéciale, on peut utiliser le même atome comme variable et étiquette à la fois sans risque de confusion. L'utilisation du PROG permet de ne pas rendre les récursions trop profondes. Reste que l'usage de cette fonction augmente légèrement (en raison des initialisations : liaisons des variables locales, liaison des étiquettes aux instructions) le temps d'execution.
Si un go, goto *ou* return *est exécutée dans une fonction qui n'est pas un* PROG, *les* go, goto *ou* return, *rentreront dans le (ou sortiront du, dans le cas de RETURN)* dernier PROG *entamé. On voit ici le moyen de sortir d'un* MAP *ou* MAPC, *ou d'une récursive,* avant *l'achèvement du déroulement complet de ces fonctions.*
Mais dans tous les cas on doit executer les suites d'instructions de façon purement séquentielle (sans boucle), on aura tout intérêt à faire plutôt usage de la fonction PROGN *ci-dessous.*

(PROGN X Y ... Z) — *progn* évalue chacun des arguments en séquence, et ramène pour valeur, la
- SUBR - N — valeur du *dernier* argument évalué..

La distinction entre PROG *et* PROGN *est assez semblable à celle qu'ALGOL fait entre* bloc *(avec déclarations initiales)et* instruction composée *(sans déclarations). Cependant l'uniformisation de LISP sous forme de fonctions permet d'attribuer une* valeur globale *à ces évaluations groupées, qu'on retrouve sous forme analogue dans ALGOL 68* [12].

(GO X) — *go* est la fonction de saut en LISP, dans un *PROG*. La prochaine évaluation
- FSUBR — exécutée sera alors celle de l'instruction étiquetée X.

(GOTO X) — semblable à *go* mais l'agent X est évalué (la valeur devant être alors une éti-
- SUBR - 1 — quette). Cette fonction sert principalement à définir les aiguillages.

(RETURN X) — un *return* est la sortie normale pour u n *prog*. Son argument est évalué, et
- SUBR - 1 — sera la valeur du *prog* en question.

(SET X Y) — Cette fonction donne à X la valeur Y. X et Y sont tous deux évalués avant
- SUBR - 2 — l'appel. La valeur de *set* est Y. La valeur de X doit obligatoirement être ou bien une variable locale d'un *PROG*, ou bien un argument formel de la fonction dans laquelle *set* est évalué, ou d'une fonction *appelant* cette fonction à un niveau supérieur. Dans tous les cas, la valeur de X devra être un atome litteral. En aucun cas *set* ou *setq* ne modifie la C-VALEUR d'un atome, mais toujours sa A-valeur.

EX: si la valeur de X est C et la valeur de Y est 5, (SET X Y) donne à C la valeur 5, et retourne la valeur 5.

On a ici une instruction équivalente dans une certainemesure à l'affectation ALGOL. On en mesure cependant la plus grande puissance par cet exemple, qu'ALGOL ne permettrait certes pas

SI *Condition* alors ***X*** sinon ***Y*** *:= expression*

```
(SET (COND
          ( ( condition ) ( QUOTE  X ) )
          ( T ( QUOTE  Y ) ) )
      expression )
```

Il faut noter que LISP est pratiquement contemporain d'ALGOL 60 [13] .

(SETQ X Y) = (SET (QUOTE X) Y)
- FSUBR — Identique à *set*, mais le premier argument n'est pas évalué.

EX : si la valeur de X est C et la valeur de Y est (A D) (SETQ X Y) donne à X la valeur (A D), ne modifie pas la valeur de C, et retourne (A D).

Dans un PROG, *le programme de trace* TRASET, *permet, pour chaque évaluation de* setq *de faire imprimer, le nom de la variable affectée, et sa nouvelle valeur par* setq.

PREDICATS ET CONNECTEURS LOGIQUES

(ATOM X) - SUBR - 1	de valeur T si X est un atome, NIL sinon.

Cette entorse (valeur T) à la règle : « tout ce qui n'est pas NIL est vrai *» s'explique par l'appel que voici :* SI *(ATOM NIL) ramenanit la valeur de son agent, en cas de satisfaction du prédicat, il ramènerait NIL, i.e.* faux *, ce qui est manifestement faux, NIL étant un atome.*

(NUMBP X) - SUBR - 1		de valeur X si X est un *nombre*, NIL sinon.
(EQ X Y) - SUBR - 2		de valeur T si X et Y sont le même atome (littéral ou numérique), NIL sinon
(NULL X) - SUBR - 1	=	(EQ X NIL)
(NOT X) - SUBR - 1		identique à NIL.
(EQUAL X Y) - SUBR - 2		de valeur T si X et Y sont identiques à *l'impression*, NIL sinon. X et Y sont alors des expressions LISP quelconques
(NEQUAL X Y) - SUBR - 2	=	(NOT (EQUAL X Y))
(AND X1 X2 ...Xn) - FSUBR		Prend un nombre quelconque d'arguments (zéro inclus). Si tous les arguments ont une valeur non nulle, sa valeur est celle de son dernier argument Xn, sinon NIL. On notera que (AND NIL) = T. L'évaluation, effectuée en séquence (le lecteur fera la comparaison avec PROGN) stoppe au premier argument dont la valeur est NIL.
(OR X1 X2 .... Xn)		Prends un nombre quelconque d'atguments (zéro inclus). Sa valeur est celle du premier argument dont la valeur n'est pas NIL, sinon NIL si tous les arguments ont la valeur NIL. On notera que (OR()) = NIL. L'évaluation cesse au premier argument dont la valeur n'est pas NIL.

On notera que AND *tout aussi bien que* OR *sont non-commutatifs, en raison de l'évaluation séquentielle. En voici un exemple frappant : la valeur de (AND (Setq X Y) (GO L)) sera à l'évidence fort différente de (AND (GO L) (Setq X Y)). Dans le second cas, le saut vers L s'effectuera immédiatement, et (SETQ X Y) ne sera jamais évalué. Dans le premier cas, si la valeur de Y est NIL, le saut vers L n'aura pas lieu.*

On utilisera avantageusement AND *à la place de* COND *dans certains cas :*

ex : SI *P1* alors *si P2* alors *e ; expression ici non - ambigüe, s'écrira (AND P1 P2 e). On économisera dans les gros programmes un certain nombre de doublets, au prix d'un plus grand soin dans l'écriture.*

(MEMBER X Y) — SUBR - 2

détermine si X est ou non membre de la liste Y, i.e. s'il existe un élément de Y *equal* à X. Si oui, sa valeur est la partie de la liste Y commençant par l'élément X, sinon sa valeur est NIL.

Ex : (MEMBER (QUOTE C) (QUOTE (A B C D E))) = (C D E)

MANIPULATION DE LISTES
ET CONCATENATION

LIST,NCONC,LENGTH,PAIR

(LIST X1 X2 ... XN) -SUBR-N — prend un nombre quelconque d'arguments tous évalués avant l'appel. Sa valeur est la liste des valeurs de ses arguments.

(NCONC X Y) -SUBR-2 — concatène physiquement les listes X et Y, i.e. le *CDR* du dernier élément de X pointe sur Y. Sa valeur est la liste ainsi obtenue.

ex : si X = (1 2) et Y = (A B)

(NCONC X Y) = (1 2 A B)

(NCONC X NIL) = (1 2)

(NCONC (LIST 9) Y) = (9 A B)

*** On notera l'utilité de NCONC pour construire des listes circulaires.*

Ex ; à partir de X = (A B) on obtiendra la liste (A B A B A .....

(NCONC X X)

On prendra le plus grand soin dans l'écriture de cette fonction qui modifie physiquement et définitivement les structures de listes. On s'attendra, lors de la mise au point, à quelques effet étranges.

(LENGTH X) -SUBR-1 — sa valeur est la longueur (nombre d'éléments) de la liste X. On peut également définir la longueur comme le nombre de CDR requis pour atteindre une liste vide.

(PAIR X Y) -SUBR-2 — Si X = (X1 X2 ... XN) et Y = (Y1 Y2 ... YN)

(PAIR X Y) construit et aura comme valeur la liste

((X1 . Y1)(X2 . Y2) ... (Xn . Yn))

C'est l'équivalent interne, en code machine de cette fonction, qui effectue la liaison des variables sur la A-liste, et celle des étiquettes dans les PROG.

X et Y peuvent être de longueur différentes.

Ex; X = (A B C), Y = (D E)

(PAIR X Y) = ((C . NIL)(B . E)(A . D))

(PAIR Y X) = ((B . E)(A . D))

X = (A B . C), Y = (D E F G)

(PAIR X Y) = ((C . (F G))(B . E)(A . D))

FONCTIONS ARITHMETIQUES

ADD1, SUB1, ZEROP, TIMES, QUO, REM, PLUS, DIFFER, GT, LT.

LISP 510 ne peut traiter que des nombres sous forme entière.

Les atomes numériques auront la représentation externe classique, i.e., signe ou non, suivi d'une chaîne dechiffres. Les atomes numériques ne possèdent pas de P-liste et sont stockés directement dans les listes dont ils sont membres. Ils occupent un mot, et, à l'évaluation ramènent leur propre valeur (i.e. pointent implicitement sur eux-mêmes) moins une constante. Par suite, faute de P-name stocké, le même nombre peut exister simultanément à différents endroits de la mémoire. (contrairement aux atomes littéraux).

Les atomes numériques peuvent être créés par READ, *ou à la suite de l'évaluation d'une fonction arithmétique. Par suite du mécanisme d'évaluation, un atome numérique ne saurait avoir ni de A-valeur ni de C-valeur, mais possède ce que l'on pourrait nommer une self-valeur. Leur « PNAME externe » n'existera que provisoirement à l'évaluation de* PRINT *ou* PRIN1.

On prendra garde à ne pas faire usage d'entiers inférieurs à - 50 000 ou supérieurs à + 50 000.

(ADD1 X) - SUBR - 1	X + 1
(SUB1 X) - SUBR - 1	X – 1
(ZEROP X) - SUBR - 1	= (EQ X 0)
(TIMES X1 X2 ... Xn) - SUBR - N	prend un nombre quelconque d'arguments. Sa valeur est le produit de X1, X2, ..., Xn.
(QUO X Y) - SUBR - 2	X / Y tronqué *ex :* (QUO 3 2) = 1 (QUO -3 2) = -1
(REM X Y) - SUBR - 2	le reste de la division *quo* de X par Y.

De plus, l'évaluation : (CDR (QUOTE REM)) livre le nombre de doublets en liste libre dégagés par le dernier garbage collecting.

(PLUS X1 X2 ... Xn) .. - SUBR - N	Prends un nombre quelconque d'arguments. Sa valeur est la somme de X1, X2, ... Xn.
(DIFFER X Y) - SUBR - 2	X – Y
(GT X Y) - SUBR - 2	T si X > Y ; NIL sinon
(LT X Y) - SUBR - 2	T si X < Y ; NIL sinon